ISE SRA

Leibniz Universität Hannover

Hardware-Praktikum

Miniprojekt: Minimax-Maschine

Paul Aumann

06.12.2021

Organisatorisches

- Minimax Projekt im Rahmen des Hardwarepraktikums

- Stud.IP Veranstaltung „Miniprojekt: Minimax-Maschine“
  - Aufgabenstellung
  - Minimax Simulator

- Ansprechpartner
  - **Paul Aumann**
    paul.aumann@gmail.com
    paul.aumann@stud.uni-hannover.de
  - apl. Prof. Dr.-Ing. habil. Jürgen Brehm
    brehm@sra.uni-hannover.de

Organisatorisches

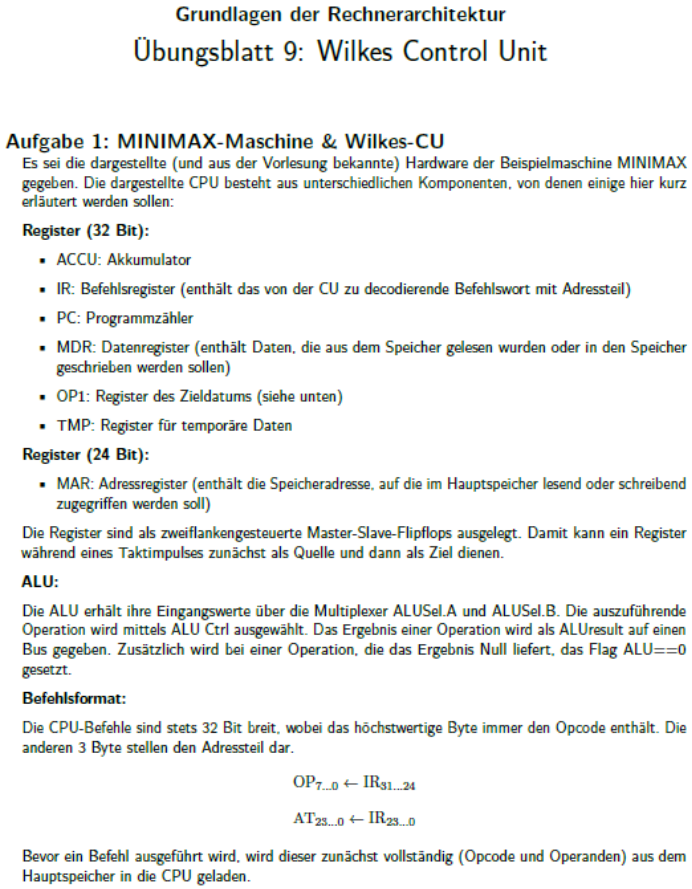
- Zur Bearbeitung
  - Es wird in 3er Gruppen gearbeitet
  - Jedes Gruppenmitglied muss den Ablauf der Minimax-Maschine erklären können
  - Eine Dokumentation mit Steuersignaltabelle und beschreibendem Text muss am Ende der Bearbeitung mit abgegeben werden
- Zwischentermin
  - Flexibel zwischen 06.01.2022 – 17.01.2022
    Je Gruppe wird es einen separaten Slot von 30 min geben
  - Ziel ist, den aktuellen Stand der Bearbeitung vorzustellen

Organisatorisches

- Abgabe
  - Die Abgabe erfolgt bis zum 31.01.2022 um 18:00 UTC+1 per E-Mail an mich
  - Spätere Abgaben werden nicht berücksichtigt
  - Die Abgabe beinhaltet den gesicherten Stand der Minimax-Maschine, die Dokumentation und das Resultat der Minimax-Maschine
  - In der Woche danach wird es dann noch mal ein Abschlusstreffen geben
- Sprechstunden
  - Sprechstunden können unter Stud.IP Sprechstunden gebucht werden ***Bitte nicht 5 vor 12 einen Termin buchen, schön wäre es mindestens 12 Stunden Vorlauf zu haben ;-)***
  - Sprechstunden sind wöchentlich verfügbar
  - Jede Sprechstunde dauert 30 min
  - Es ist möglich, mir vorab per E-Mail die Fragen oder den Code zu übermitteln

Minimax-Maschine

- Bekannt aus Vorlesung „Grundlagen der Rechenarchitektur“
- Sie besteht aus
  - Register
  - Speicher
  - Multiplexer
  - Alu
- Die Funktion des Programms wird beschrieben durch
  - Flussdiagramm
  - Steuersignaltabelle



Grundlagen der Rechnerarchitektur

Übungsblatt 9: Wilkes Control Unit

Aufgabe 1: MINIMAX-Maschine & Wilkes-CU

Es sei die dargestellte (und aus der Vorlesung bekannte) Hardware der Beispielmaschine MINIMAX gegeben. Die dargestellte CPU besteht aus unterschiedlichen Komponenten, von denen einige hier kurz erläutert werden sollen:

Register (32 Bit):

- ACCU: Akkumulator
- IR: Befehlsregister (enthält das von der CU zu decodierende Befehlswort mit Adressteil)
- PC: Programmzähler
- MDR: Datenregister (enthält Daten, die aus dem Speicher gelesen wurden oder in den Speicher geschrieben werden sollen)
- OP1: Register des Zieldatums (siehe unten)
- TMP: Register für temporäre Daten

Register (24 Bit):

- MAR: Adressregister (enthält die Speicheradresse, auf die im Hauptspeicher lesend oder schreibend zugegriffen werden soll)

Die Register sind als zweiflankengesteuerte Master-Slave-Flipflops ausgelegt. Damit kann ein Register während eines Taktimpulses zunächst als Quelle und dann als Ziel dienen.

ALU:

Die ALU erhält ihre Eingangswerte über die Multiplexer ALUSel.A und ALUSel.B. Die auszuführende Operation wird mittels ALU Ctrl ausgewählt. Das Ergebnis einer Operation wird als ALUresult auf einen Bus gegeben. Zusätzlich wird bei einer Operation, die das Ergebnis Null liefert, das Flag ALU==0 gesetzt.

Befehlsformat:

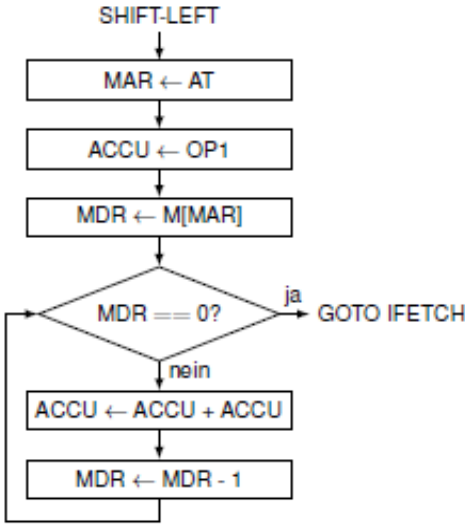
Die CPU-Befehle sind stets 32 Bit breit, wobei das höchstwertige Byte immer den Opcode enthält. Die anderen 3 Byte stellen den Adressteil dar.

$$OP_{7...0} \leftarrow IR_{31...24}$$

$$AT_{23...0} \leftarrow IR_{23...0}$$

Bevor ein Befehl ausgeführt wird, wird dieser zunächst vollständig (Opcode und Operanden) aus dem Hauptspeicher in die CPU geladen.

Minimax-Maschine

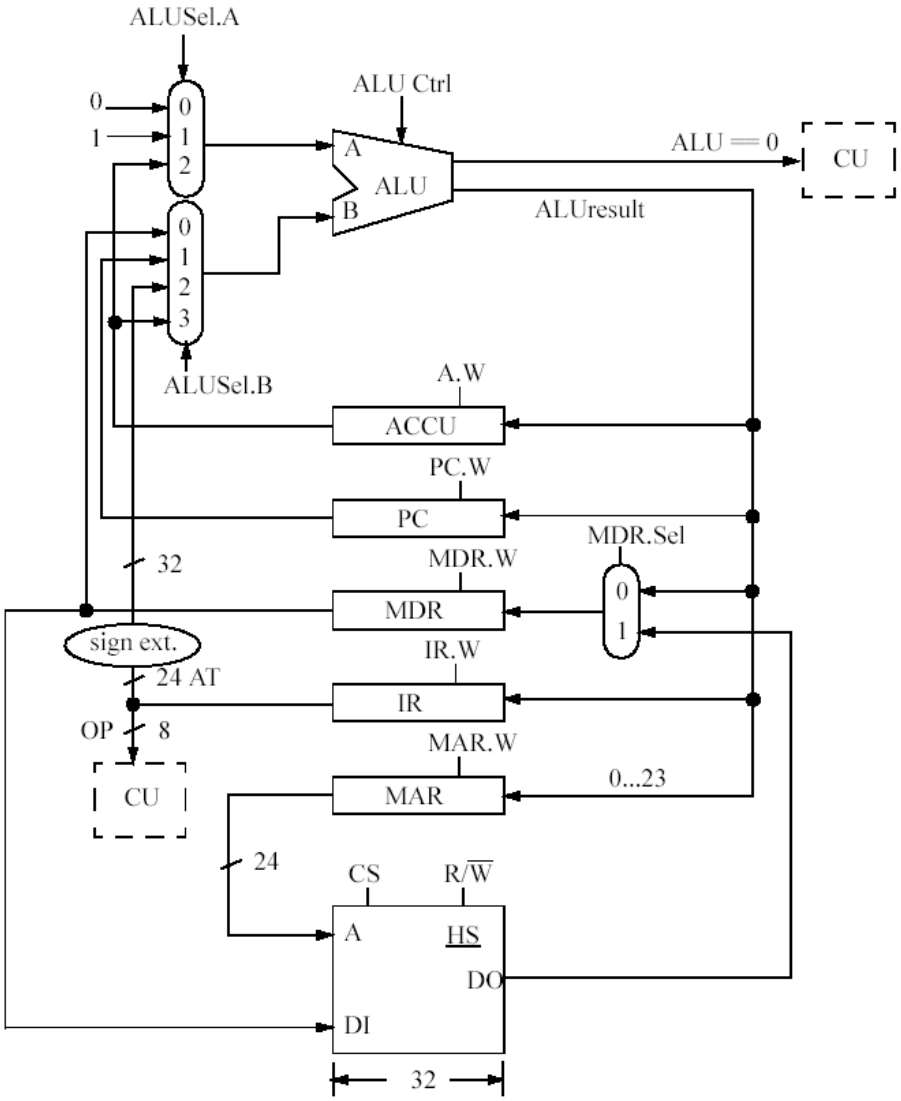


Label	Adresse CM	ALUSel.A	ALUSel.B	MDR.Sel	HS CS	HS R/$\overline{W}$	ALU Ctrl	OP1.W	TMP.W	ACCU.W	PC.W	MDR.W	IR.W	MAR.W	GOTO OP	Cond ALU==0	A	Bemerkungen
Signalnr.		c_0 c_1	c_2 c_3 c_4	c_5	c_6	c_7	c_8 c_9 c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_{17}	c_{18}			
IFETCH:	0	x x	0 0 0	x	0	x	0 1 0	0	0	0	0	0	0	1	0	x	1	MAR ← PC
	1	x x	x x x	1	1	1	x x x	0	0	0	0	1	0	0	0	x	2	MDR ← M[MAR]
	2	0 1	0 0 0	x	0	x	0 0 0	0	0	0	1	0	0	0	0	x	3	PC ← PC +1
	3	x x	0 1 0	x	0	x	0 1 0	0	0	0	0	0	1	0	0	x	4	IR ← MDR
	4	x x	x x x	x	x	x	x x x	x	x	x	x	x	x	x	1	x	5	GOTO OP
SHIFT-LEFT:	5	x x	0 1 1	x	0	x	0 1 0	0	0	0	0	0	0	1	0	x	6	MAR ← AT
	6	0 0	x x x	x	0	x	0 0 1	0	0	1	0	0	0	0	0	x	7	ACCU ← OP1
	7	x x	x x x	1	1	1	x x x	0	0	0	0	1	0	0	0	x	8	MDR ← M[MAR]
	8	x x	0 1 0	x	0	x	0 1 0	0	0	0	0	0	0	0	0	1 0	0 9	GOTO IFETCH
loop:	9	1 0	0 0 1	x	0	x	0 0 0	0	0	1	0	0	0	0	0	x	10	ACCU ← ACCU + ACCU
	10	0 1	0 1 0	0	0	x	1 0 0	0	0	0	0	1	0	0	0	1 0	0 9	GOTO IFETCH GOTO loop MDR ← MDR - 1
	11																	
	12																	
	13																	

Minimax-Basis-Maschine

- **Register (32 Bit):**
  - ACCU: Akkumulator
  - IR: Befehlsregister (enthält das von der CU zu decodierende Befehlswort mit Adressteil)
  - PC: Programmzähler
  - MDR: Datenregister (enthält Daten, die aus dem Speicher gelesen wurden oder in den Speicher geschrieben werden sollen)
- **Register (24 Bit):**
  - MAR: Adressregister (enthält die Speicheradresse, auf die im Hauptspeicher lesend oder schreibend zugegriffen werden soll)
- **ALU:**
  - Die ALU erhält ihre Eingangswerte über die Multiplexer ALUSel.A und ALUSel.B. Die auszuführende Operation wird mittels ALU Ctrl ausgewählt. Das Ergebnis einer Operation wird als ALUresult auf einen Bus gegeben. Zusätzlich wird bei einer Operation, die das Ergebnis Null liefert, das Flag ALU==0 gesetzt.

Minimax-Basis-Maschine



Minimax-Basis-Maschine

- Die Minimax-Basis Maschine ist Turing-vollständig
  - Im Jahre 1954 veröffentliche Hans Hermes den Beweis, dass Von-Neumann-Rechenmaschinen Turing-vollständig sind
- Es können aber zusätzlich zur Basis Register, Operationen und Konstanten definiert werden
- Die Adressierung der Speicherbereiche erfolgt **Wortbasiert**
  - Jede Adresse des Hauptspeichers liefert 4 Byte an die Minimax-Maschine

Die Aufgabe

- Entschlüsseln Sie einen Datenbereich auf dem Hauptspeicher, der nach einer Ransomware-Attacke nicht mehr zu lesen ist

Die Aufgabe

- Folgende Informationen sind Ihnen bekannt
  - Sie kennen die Adresse, in der sich die verschlüsselte Datei befindet
  - Sie kennen den Verschlüsselungsalgorithmus „RC4“
  - Sie haben nach Durchsuchen des Hauptspeichers den Key gefunden
  - Sie wissen, dass die gesuchte Datei vom Format JPG ist
- Folgende Aufgabe wird an Sie gestellt
  - Sie müssen mit Hilfe der Minimax-Basis Maschine den verschlüsselten Text wieder lesbar machen
  - Sie können dafür die Minimax-Basis Maschine erweitern

Die Aufgabe – Competition (nicht Pflicht)

- Am Ende der Bearbeitungszeit wird der effektivste Algorithmus mit einem Preis prämiert
- Der Haken: ALU-Operationen kosten Punkte
  - Für jede Erweiterung der Basis Maschine werden Punkte berechnet
  - Jedes ergänztes Register, jede zusätzliche Konstante bedeutet **1 Punkt**
  - Jede zusätzliche Std. Operation (Sub,Add,Inc,Dec) bedeutet **4 Punkte**
  - Jede zusätzliche Operation (Mul,Div,Mod) bedeutet **10 Punkte**
  - Jede zusätzliche Operation (OR,AND,XOR,INV) bedeutet **8 Punkte**
  - Jede zusätzliche Operation (Shift*) bedeutet **5 Punkte**
  - Jede zusätzliche Operation (Rotate*) bedeutet **7 Punkte**
- Der zweite Haken: Codelänge
  - Die Länge des Algorithmus wird mit **0,5 Punkte** pro Zeile bewertet
- Der Algorithmus mit den wenigsten Punkten gewinnt

RC4 Algorithmus

- RC4 (Rivest Cipher 4)
- Stream Cipher
- Symmetric key algorithm
- Eingesetzt bei: Https, SSH1, WEP bzw. WPA

- Der Algorithmus setzt sich aus zwei Teilen zusammen
  - S-Box (substitution-box)
  - PRGA (Pseudo-random generation algorithm)

RC4 Algorithmus

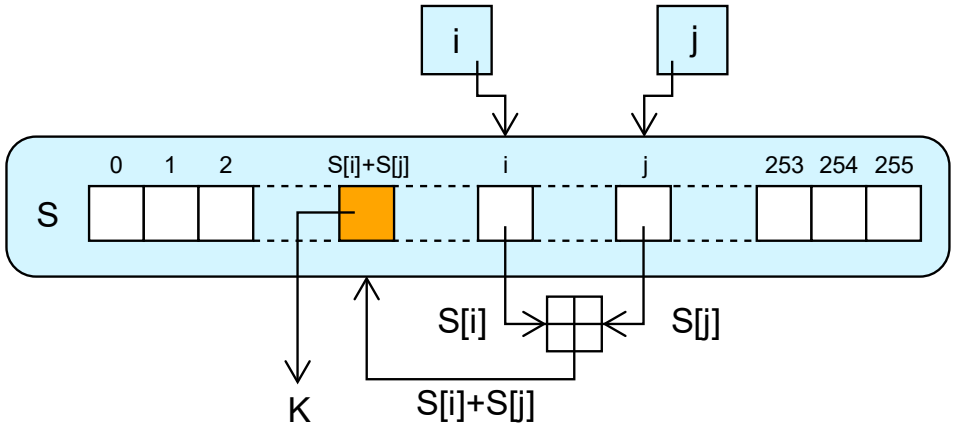
- S-Box
  - Eine Permutation über alle 256 möglichen Bytes
- Die S-Box wird initial mit den Bytes 0 - 255 gefüllt (bereits in StudIP vorhanden)
- Danach wird sie mit Hilfe des Keys permutiert

```
for i from 0 to 255
    S[i] := i
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap values of S[i] and S[j]
endfor
```

RC4 Algorithmus

- PRGA
  - Für beliebig viele Iterationen modifiziert der PRGA den Zustand und gibt ein Byte des Keystreams aus

```
i := 0
j := 0
x := 0
z[] := text
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    K := S[(S[i] + S[j]) mod 256]
    z[x] ^= K
    x := x + 1
endwhile
```



Bitoperation

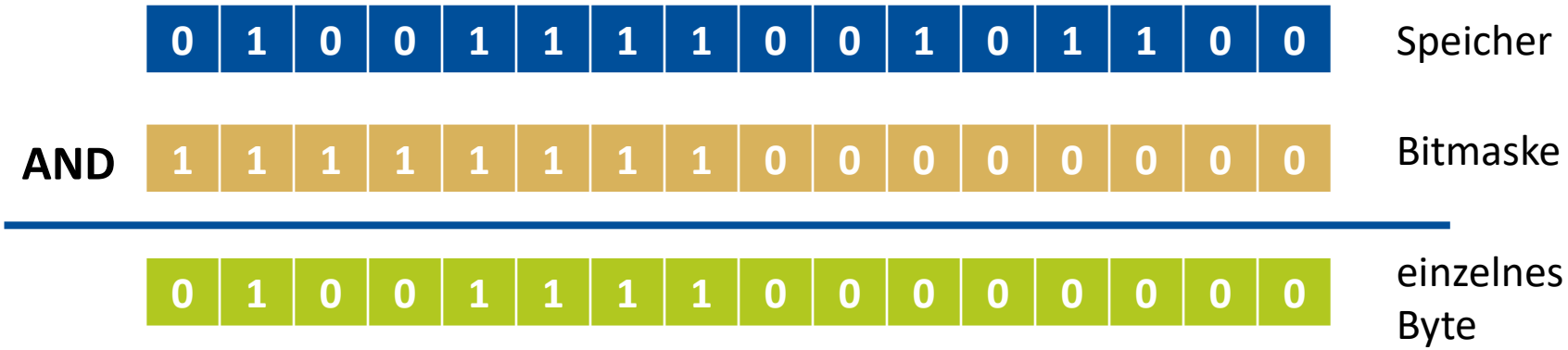
- Bitoperationen können genutzt werden, um in einem Bytestream einzelne Bytes zu separieren und manipulieren

- Folgende Bitoperationen gibt es
  - NICHT
  - ODER
  - XOR
  - UND

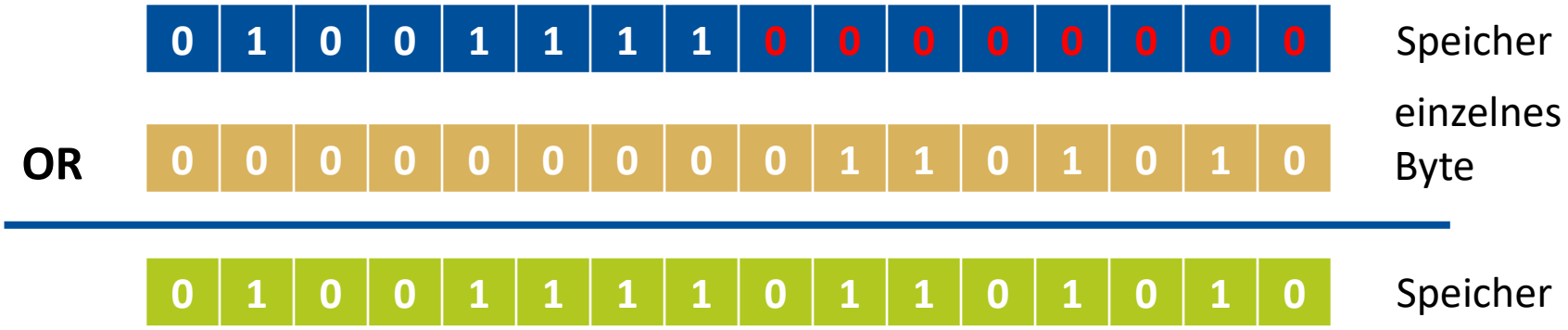
	0101	0101	0101
NICHT 0111	ODER 0011	UND 0011	XOR 0011
= 1000	= 0111	= 0001	= 0110

Bitoperation Maskierung

Byte lesen

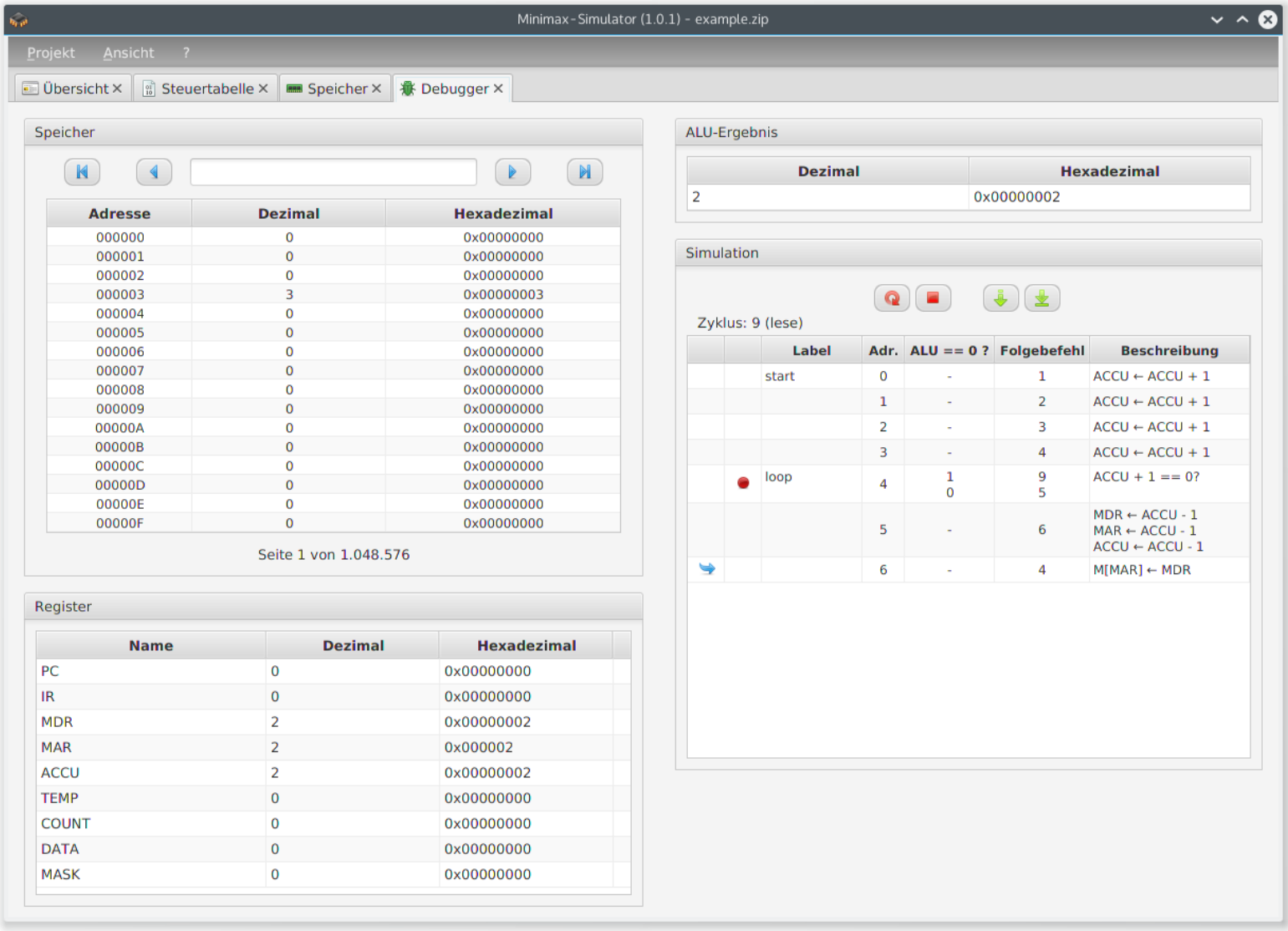


Byte schreiben



Das zu schreibende Byte im Speicher muss vorher mit Nullen gelöscht werden

Der Minimax-Simulator

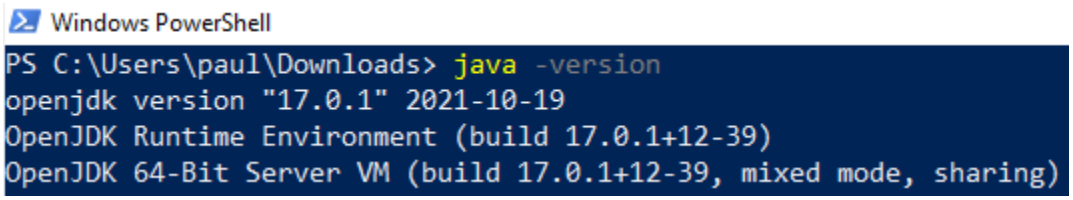


Der Minimax-Simulator

- Plattformunabhängiger Simulator für Minimax-Maschinen
- Auf Stud.IP bereitgestellt
- Open Source Projekt auf GitHub
  - https://github.com/luhsra/MinimaxSimulator
- Weitere Dokumentation
  - https://luhsra.github.io/MinimaxSimulator/
- Systemvoraussetzungen
  - Java 8u40

Installation unter Windows (Windows 10)

- Download und Installation von openjdk 17 https://jdk.java.net/17/
  - `java -version` liefert:



- Ausführung des Simulators mittels
  `java -jar .\minimax_simulator-2.0.0-jar-with-dependencies.jar`

Installation unter Linux (Ubuntu 20.04)

- Installation von Java mittels `sudo apt-get install openjdk-17-jre`
  - `java –version` liefert:

```
paul@paul:~/Downloads$ java -version
openjdk version "17" 2021-09-14
OpenJDK Runtime Environment (build 17+35-Ubuntu-120.04)
OpenJDK 64-Bit Server VM (build 17+35-Ubuntu-120.04, mixed mode, sharing)
```

- Download und Entpacken von javafx-17.0.1 unter https://gluonhq.com/products/javafx/

- Ausführung des Simulator mittels

```
java -jar --module-path <javafx-path>/javafx-sdk-17.0.1/lib –add-modules javafx.controls,javafx.fxml minimax_simulator-2.0.0-jar-with-dependencies.jar
```

Installation unter macOS (Big Sur 11.0.1)

- Installation von openjdk 17 https://jdk.java.net/17/
  - `java –version` liefert:

```
paul@Pauls-iMac ~ % java -version
openjdk version "17.0.1" 2021-10-19
OpenJDK Runtime Environment (build 17.0.1+12-39)
OpenJDK 64-Bit Server VM (build 17.0.1+12-39, mixed mode, sharing)
```

- Download und Entpacken von javafx-17.0.1 unter https://gluonhq.com/products/javafx/
  - (https://download2.gluonhq.com/openjfx/17.0.1/openjfx-17.0.1_osx-x64_bin-sdk.zip direkter Link für macOS x64)
- Ausführung des Simulator mittels

```
java -jar --module-path <javafx-path>/javafx-sdk-17.0.1/lib –add-modules javafx.controls,javafx.fxml minimax_simulator-2.0.0-jar-with-dependencies.jar
```

Vorhandene Dateien in StudIP

- `key`
  Diese Datei beinhaltet den Key zum Entschlüsseln
- `data_encrypted`
  Die zu entschlüsselnde Datei
- `sBox`
  Vorkonfigurierte sBox mit den Werten 0 bis 255
  !Achtung! Jede Speicherstelle beinhaltet 4 Byte

- `sBox_swap_test_key`
  Test Key zum Überprüfen der Permutation
  Wenn Sie diesen Key verwenden um die sBox zu permutieren, sollte am Ende die `sBox_swap_test` im Speicher stehen
- `sBox_swap_test`
  Ergebnis nach SBox Permutation mit `sBox_swap_test_key`